Jack Girard
ECE 637
Lab 1: Image Filtering

1. C Programming

No Deliverables

2. Displaying and Exporting Images in Python

No Deliverables

3. FIR Low Pass Filter

1. Derivation of $H\left(e^{j\mu}, e^{jv}\right)$: *See Next Page*

Jack Girard
ECE637
Lab 1 Work

$$h(m,n) = \begin{cases} 1/81 & \text{for } |m| \le 4 \text{ and } |n| \le 4 \\ 0 & \text{otherwise} \end{cases}$$

Identity Used:

$$sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

Apply formula for DSFT & Simplify:

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \nu n)}$$

$$= \sum_{n=-4}^{4} \sum_{m=-4}^{4} \frac{1}{81} e^{-j(\mu m + \nu n)}$$

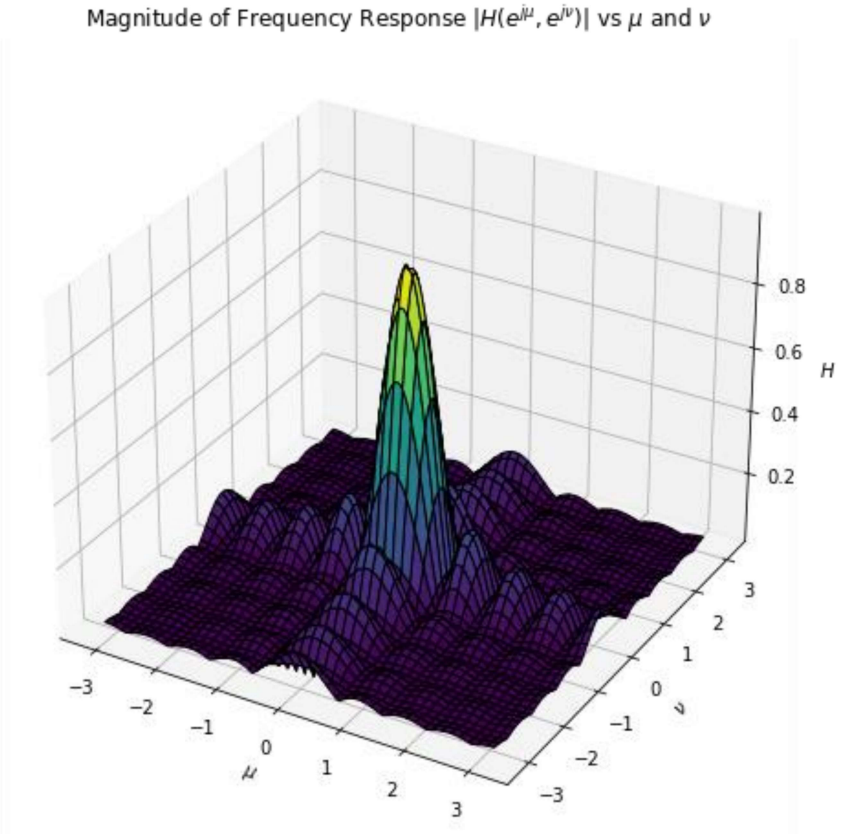
$$= \frac{1}{81} \sum_{n=-4}^{4} e^{-j\mu m} \sum_{m=-4}^{4} e^{-j\nu n}$$

$$= \frac{1}{81} \left[\left(1 + e^{-4\mu} + e^{4\mu} + e^{-3\mu} + e^{3\mu} + e^{-2\mu} + e^{2\mu} + e^{-\mu} + e^{\mu}\right) \cdots \left(1 + e^{-4\nu} + e^{4\nu} + e^{-3\nu} + e^{3\nu} + e^{-2\nu} + e^{2\nu} + e^{-\nu} + e^{\nu}\right) \right]$$

$$= \frac{1}{81} \left[\left(1 + 2\cos(4\mu) + 2\cos(3\mu) + 2\cos(2\mu) + 2\cos(\mu)\right) \cdots \left(1 + 2\cos(4\nu) + 2\cos(3\nu) + 2\cos(2\nu) + 2\cos(\nu)\right) \right]$$

$$= \boxed{\frac{1}{81} \left[\left(1 + 2\sum_{k=1}^{4} \cos(k\mu)\right) \left(1 + 2\sum_{l=1}^{4} \cos(l\nu)\right) \right]}$$

2. Plot of $\left|H\left(e^{j\mu}, e^{j\nu}\right)\right|$



3. Color image *img03.tif*

4. The filtered image *color.tif*

5. Listing of C Code: *See Next Page*

```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);
// initialize applyFilter function
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double** usedFilter, int PSF_dim);

int main (int argc, char **argv)
{
    FILE* fp;
    struct TIFF_img input_img, color_img;

    if ( argc != 2 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen(argv[1], "rb") ) == NULL) {
        fprintf( stderr, "cannot open file %s\n", argv[1] );
        exit( 1 );
    }

    /* read image */
    if ( read_TIFF( fp, &input_img ) ) {
        fprintf( stderr, "error reading file %s\n", argv[1] );
        exit( 1 );
    }

    /* close image file */
    fclose( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
        fprintf( stderr, "error:  image must be 24-bit color\n" );
        exit( 1 );
    }

    // declare 1D array of double pointers filter
    double* filter[9];

    // iterate through 1D array filter and allocate enough memory for 9 doubles each
    for (int i = 0; i < 9; i++) {
        filter[i] = malloc(sizeof(double) * 9);
```

```
48      }
49      // populate filter array according to h(m,n)
50      for (int i = 0; i < 9; i++) {
51          for (int j = 0; j < 9; j++) {
52              filter[i][j] = 1.0 / 81.0;
53          }
54      }
55
56      /* set up structure for output color image */
57      /* Note that the type is 'c' rather than 'g' */
58      get_TIFF( &color_img, input_img.height, input_img.width, 'c' );
59
60      // declare and initialize integer to store the dimension of the point
          spread function
61      int PSF_dim = 9;
62
63      // apply filter using applyFilter function as defined below main
64      applyFilter(&color_img, &input_img, filter, PSF_dim);
65
66      /* open color image file */
67      if ( ( fp = fopen( "color.tif" , "wb" ) ) == NULL) {
68          fprintf( stderr, "cannot open file color.tif\n" );
69          exit( 1 );
70      }
71
72      /* write color image */
73      if ( write_TIFF( fp, &color_img ) ) {
74          fprintf( stderr, "error writing TIFF file %s\n", argv[2] );
75          exit( 1 );
76      }
77
78      /* close color image file */
79      fclose( fp );
80
81      /* de-allocate space which was used for the images */
82      free_TIFF( &(input_img) );
83      free_TIFF( &(color_img) );
84
85      return(0);
86  }
87
88  void error(char* name)
89  {
90      printf("usage:  %s  image.tiff \n\n", name);
91      printf("this program reads in a 24-bit color TIFF image.\n");
92      printf("It then horizontally filters the green component, adds noise,
          \n");
93      printf("and writes out the result as an 8-bit image\n");
94      printf("with the name 'green.tiff'.\n");
```

```
95      printf("It also generates an 8-bit color image,\n");
96      printf("that swaps red and green components from the input image");
97      exit(1);
98  }
99
100 // limitIntensity function definition
101 int limitIntensity(double inputValue) {
102     // declare an integer variable newValue and initialize it to zero
103     int newValue = 0;
104     // if input value parameter is less than zero, assign new value to 0
105     if (inputValue < 0) {
106         newValue = 0;
107     }
108     // if input value parameter is greater than 255, assign new value to
        255
109     else if (inputValue > 255) {
110         newValue = 255;
111     }
112     // otherwise, assign new value to the input value parameter re-cast as
        an integer
113     else {
114         newValue = (int)inputValue;
115     }
116     return newValue;
117 }
118
119 // applyFilter function definition
120 void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img,
      double** usedFilter, int PSF_dim) {
121     // declare and define N - the dimension of the point spread function
        (PSF)
122     int N = (PSF_dim - 1) / 2;
123     // declare and define image height and width based on input image TIFF
        struct methods
124     int img_height = input_img->height;
125     int img_width = input_img->width;
126     // declare doubles to store red, green, and blue value for each pixel
127     double redPlane, greenPlane, bluePlane;
128     // declare PSF variables
129     int m, n;
130     // declare variables to store current location within PSF
131     int a, b;
132     // for each pixel:
133     for (int i = 0; i < img_height; i++) {
134         for (int j = 0; j < img_width; j++) {
135             // initialize RGB values to zero
136             redPlane = 0.0;
137             greenPlane = 0.0;
138             bluePlane = 0.0;
```

```
139             // for each pixel in the PSF (9*9 in this case)
140             for (m = -N; m <= N; m++) {
141                 for (n = -N; n <= N; n++) {
142                     // assign a and b to current PSF matrix location
143                     a = i - m;
144                     b = j - n;
145                     // if a and b are within the image boundaries
146                     if (a >= 0 && a < img_height && b >= 0 && b <
                    img_width) {
147                         // apply filter by summing across PSF according to
                     difference equation for 2D filters
148                         redPlane += usedFilter[m + N][n + N] * input_img-
                    >color[0][a][b];
149                         greenPlane += usedFilter[m + N][n + N] *
                    input_img->color[1][a][b];
150                         bluePlane += usedFilter[m + N][n + N] * input_img-
                    >color[2][a][b];
151                     }
152                 }
153             }
154             // populate output image method for color after calling
                  limitIntensity function to ensure acceptable RGB values
155             output_img->color[0][i][j] = limitIntensity(redPlane);
156             output_img->color[1][i][j] = limitIntensity(greenPlane);
157             output_img->color[2][i][j] = limitIntensity(bluePlane);
158         }
159     }
160 }
```

4. FIR Sharpening Filter

1. Derivation of $H\left(e^{j\mu}, e^{jv}\right)$: *See Next Page*
2. Derivation of $G\left(e^{j\mu}, e^{jv}\right)$: *See Next Page*

Jack Girard
ECE637
Lab 1 Work

Let $h(m,n)$ be a low pass filter. For our purposes use

$$h(m,n) = \begin{cases} 1/25 & \text{for } |m| \leq 2 \text{ and } |n| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

The unsharp mask filter is then given by

$$g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n))$$

where λ is a constant greater than zero.

Identity Used:

$$sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

$$cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$

Apply formula for DSFT & Simplify:

$$H(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(m,n) e^{-j(\mu m + \nu n)}$$

$$= \sum_{n=-2}^{2} \sum_{m=-2}^{2} \frac{1}{25} e^{-j(\mu m + \nu n)}$$

$$= \frac{1}{25} \sum_{n=-2}^{2} e^{-j\mu m} \sum_{m=-2}^{2} e^{-j\nu n}$$

$$= \frac{1}{25} \left[(1 + e^{-2\mu} + e^{2\mu} + e^{-\mu} + e^{\mu})(1 + e^{-2\nu} + e^{2\nu} + e^{-\nu} + e^{\nu}) \right]$$

$$= \frac{1}{25} \left[(1 + 2\cos(2\mu) + 2\cos(\mu))(1 + 2\cos(2\nu) + 2\cos(\nu)) \right]$$

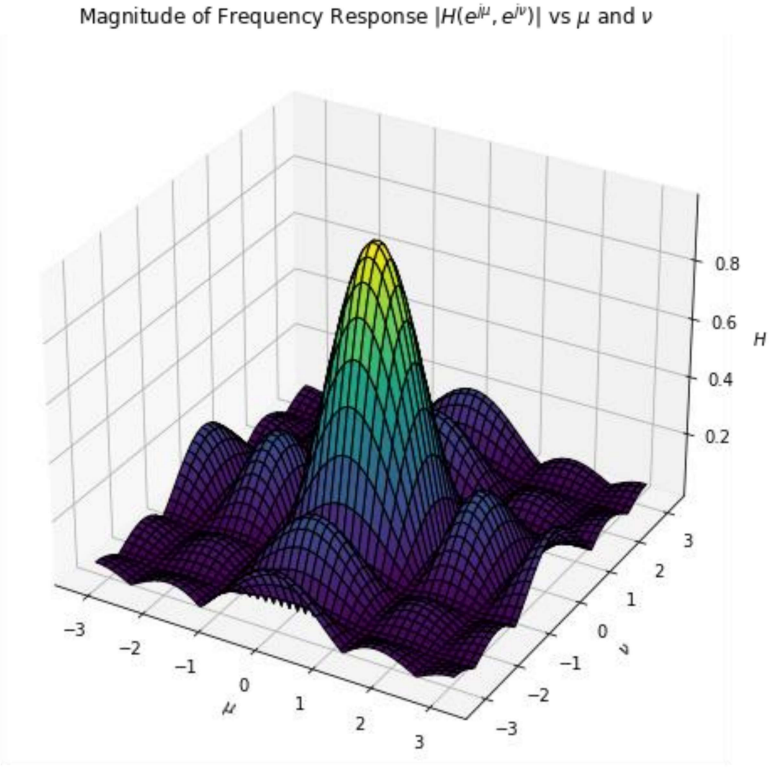
$$= \boxed{\frac{1}{25} \left[\left(1 + 2\sum_{k=1}^{2} \cos(k\mu)\right)\left(1 + 2\sum_{l=1}^{2} \cos(l\nu)\right) \right]}$$

Apply formula for DSFT & Simplify, knowing $H(e^{j\mu}, e^{j\nu})$:

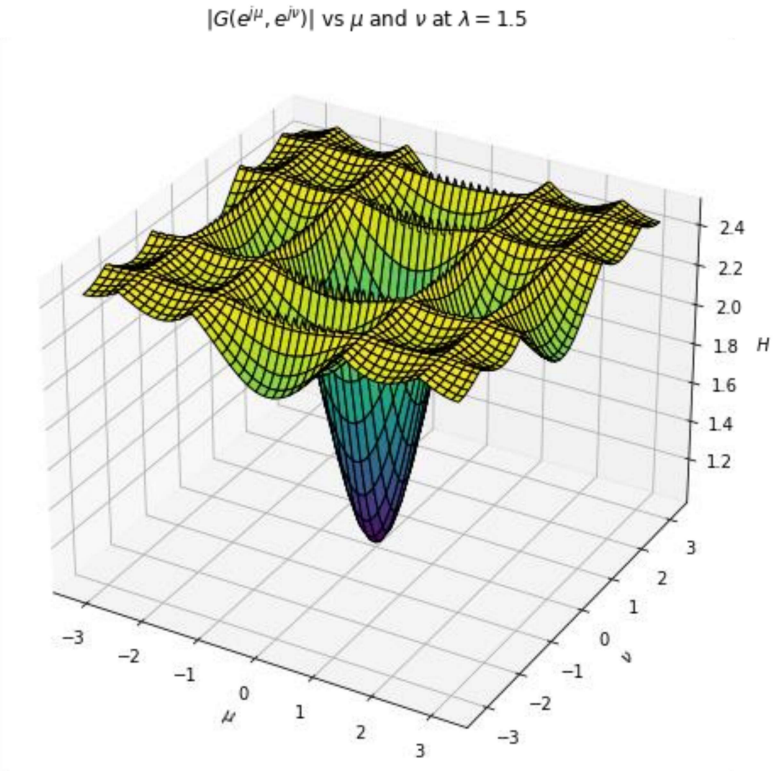
$$G(e^{j\mu}, e^{j\nu}) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} g(m,n) e^{-j(\mu m + \nu n)} = 1 + \lambda(1 - H(e^{j\mu}, e^{j\nu}))$$

$$= \boxed{1 + \lambda \left[1 - \frac{1}{25} \left[\left(1 + 2\sum_{k=1}^{2} \cos(k\mu)\right)\left(1 + 2\sum_{l=1}^{2} \cos(l\nu)\right) \right] \right]}$$

3. Plot of $\left|H\left(e^{j\mu}, e^{j\nu}\right)\right|$



4. Plot of $\left|G\left(e^{j\mu}, e^{j\nu}\right)\right|$ for $\lambda = 1.5$



5. Input color *imgblur.tif*

6. Output sharpened color image, *sharpened.tif*, for $\lambda = 1.5$

7. Listing of C Code: *See Next Page*

```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);
// initialize applyFilter function
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img, double** usedFilter, int PSF_dim, double lambda);

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, color_img;

    // adjustment: if number of arguments is greater than 3, since third will be lambda value
    if ( argc > 3 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
    }

    // declare 1D array of double pointers filter
    double* filter[5];
    // declare double to store scaling factor lambda - default value below
    double lambda = 1.0;
    if (argc > 2) {
```

```
48          lambda = atof(argv[2]);
49      }
50
51      // iterate through 1D array filter and allocate enough memory for 9
          doubles each
52      for (int i = 0; i < 5; i++) {
53          filter[i] = malloc(sizeof(double) * 9);
54      }
55      // populate filter array according to g(m,n)
56      for (int i = 0; i < 5; i++) {
57          for (int j = 0; j < 5; j++) {
58              if (i == 2 && j == 2) {
59                  filter[i][j] = 1 + lambda * (1 - 1.0 / 25.0);
60              }
61              else {
62                  filter[i][j] = lambda * (-1.0 / 25.0);
63              }
64          }
65      }
66
67      /* set up structure for output color image */
68      /* Note that the type is 'c' rather than 'g' */
69      get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
70
71      // declare and initialize integer to store the dimension of the point
          spread function
72      int PSF_dim = 5;
73
74      // apply filter using applyFilter function as defined below main
75      applyFilter(&color_img, &input_img, filter, PSF_dim, lambda);
76
77      /* open color image file */
78      if ( ( fp = fopen ( "sharpened.tif", "wb" ) ) == NULL ) {
79          fprintf ( stderr, "cannot open file color.tif\n");
80          exit ( 1 );
81      }
82
83      /* write color image */
84      if ( write_TIFF ( fp, &color_img ) ) {
85          fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
86          exit ( 1 );
87      }
88
89      /* close color image file */
90      fclose ( fp );
91
92      /* de-allocate space which was used for the images */
93      free_TIFF ( &(input_img) );
94      free_TIFF ( &(color_img) );
```

```
 95
 96      return(0);
 97  }
 98
 99  void error(char *name)
100  {
101      printf("usage:  %s  image.tiff \n\n",name);
102      printf("this program reads in a 24-bit color TIFF image.\n");
103      printf("It then horizontally filters the green component, adds noise,
           \n");
104      printf("and writes out the result as an 8-bit image\n");
105      printf("with the name 'green.tiff'.\n");
106      printf("It also generates an 8-bit color image,\n");
107      printf("that swaps red and green components from the input image");
108      exit(1);
109  }
110
111  // limitIntensity function definition
112  int limitIntensity(double inputValue) {
113      // declare an integer variable newValue and initialize it to zero
114      int newValue = 0;
115      // if input value parameter is less than zero, assign new value to 0
116      if (inputValue < 0) {
117          newValue = 0;
118      }
119      // if input value parameter is greater than 255, assign new value to
           255
120      else if(inputValue > 255) {
121          newValue = 255;
122      }
123      // otherwise, assign new value to the input value parameter re-cast as
           an integer
124      else {
125          newValue = (int)inputValue;
126      }
127      return newValue;
128  }
129
130  // applyFilter function definition
131  void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img,
       double **usedFilter, int PSF_dim, double lambda) {
132      // declare and define N - the dimension of the point spread function
           (PSF)
133      int N = (PSF_dim - 1) / 2;
134      // declare and define image height and width based on input image TIFF
           struct methods
135      int img_height = input_img->height;
136      int img_width = input_img->width;
137      // declare doubles to store red, green, and blue value for each pixel
```

```
138     double redPlane, greenPlane, bluePlane;
139     // declare PSF variables
140     int m, n;
141     // declare variables to store current location within PSF
142     int a, b;
143     // for each pixel
144     for (int i = 0; i < img_height; i++) {
145         for (int j = 0; j < img_width; j++) {
146             // initialize RGB values to zero
147             redPlane = 0.0;
148             greenPlane = 0.0;
149             bluePlane = 0.0;
150             // for each pixel in the PSF (5*5 in this case)
151             for (m = -N; m <= N; m++) {
152                 for (n = -N; n <= N; n++) {
153                     // assign a and b to current PSF matrix location
154                     a = i - m;
155                     b = j - n;
156                     // if a and b are within the image boundaries
157                     if (a >= 0 && a < img_height && b >= 0 && b <
                    img_width) {
158                         // apply filter by summing across PSF according to
                     difference equation for 2D filters
159                         redPlane += usedFilter[m + N][n + N] * input_img-
                    >color[0][a][b];
160                         greenPlane += usedFilter[m + N][n + N] *
                    input_img->color[1][a][b];
161                         bluePlane += usedFilter[m + N][n + N] * input_img-
                    >color[2][a][b];
162                     }
163                 }
164 
165             }
166             // populate output image method for color after calling
                  limitIntensity function to ensure acceptable RGB values
167             output_img->color[0][i][j] = limitIntensity(redPlane);
168             output_img->color[1][i][j] = limitIntensity(greenPlane);
169             output_img->color[2][i][j] = limitIntensity(bluePlane);
170         }
171     }
172 }
```

5. IIR Filter

1. Derivation of $H\left(e^{j\mu}, e^{jv}\right)$: *See Next Page*

Jack Girard
ECE637
Lab 1 Work

Let $h(m,n)$ be the impulse response of an IIR filter with corresponding difference equation

$$y(m,n) = 0.01x(m,n) + 0.9(y(m-1,n) + y(m,n-1)) - 0.81y(m-1,n-1)$$

where $x(m,n)$ is the input and $y(m,n)$ is the output.

Identity Used:

$$e^{i\phi} = \cos\phi + i\sin\phi$$

Taking Z-Transformation of difference equation:

$$Y(z_1,z_2) = 0.01X(z_1,z_2) + 0.9z_1^{-1}Y(z_1,z_2) + 0.9z_2^{-1}Y(z_1,z_2) - 0.81z_1^{-1}z_2^{-1}Y(z_1,z_2)$$

$$Y(z_1,z_2) - 0.9z_1^{-1}Y(z_1,z_2) - 0.9z_2^{-1}Y(z_1,z_2) + 0.81z_1^{-1}z_2^{-1}Y(z_1,z_2) = 0.01X(z_1,z_2)$$

$$Y(z_1,z_2)(1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}) = 0.01X(z_1,z_2)$$

$$H(z_1,z_2) = \frac{Y(z_1,z_2)}{X(z_1,z_2)} = \frac{0.01}{1 - 0.9z_1^{-1} - 0.9z_2^{-1} + 0.81z_1^{-1}z_2^{-1}}$$

$$H(e^{j\mu},e^{j\nu}) = \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} + 0.81e^{-j\mu}e^{-j\nu}}$$

$$\boxed{H(e^{j\mu},e^{j\nu}) = \frac{0.01}{1 - 0.9e^{-j\mu} - 0.9e^{-j\nu} + 0.81e^{-j(\mu+\nu)}}}$$

Further Simplifying $H(e^{j\mu},e^{j\nu})$ and calculating $|H(e^{j\mu},e^{j\nu})|$

$$= \frac{0.01}{1 - 0.9(\cos\mu - j\sin\mu) - 0.9(\cos\nu - j\sin\nu) + 0.81(\cos(\mu+\nu) - j\sin(\mu+\nu))}$$

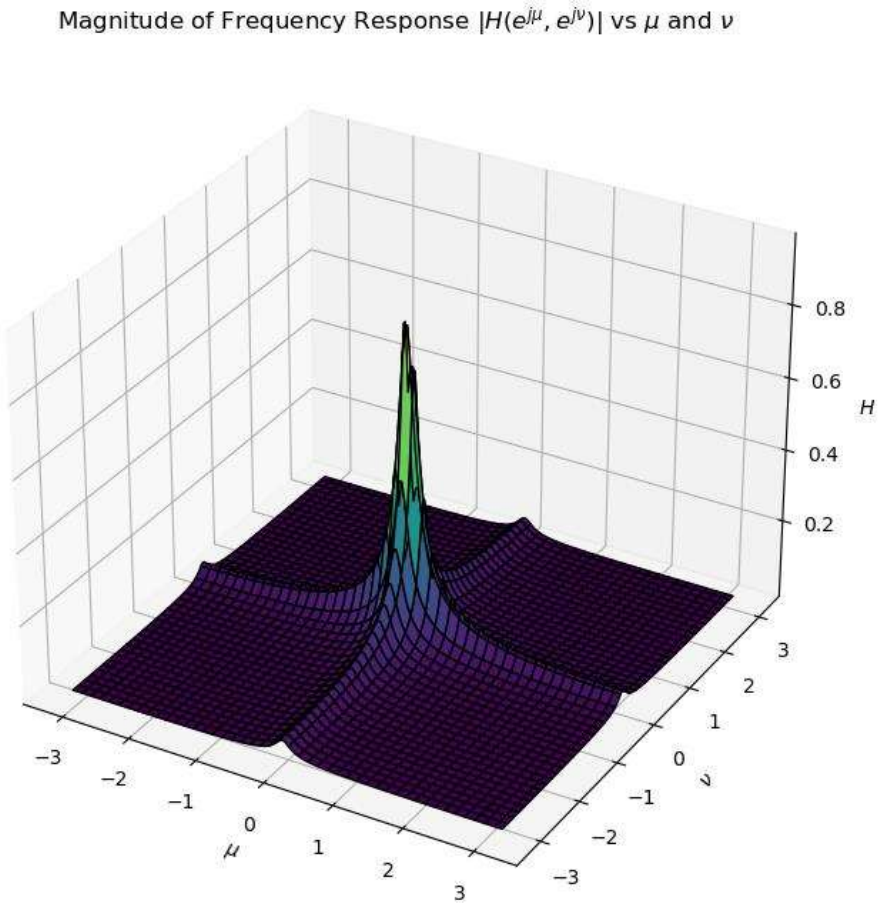
$$= \frac{0.01}{[1 - 0.9\cos\mu - 0.9\cos\nu + 0.81\cos(\mu+\nu)] + [0.9j\sin\mu + 0.9j\sin\nu - 0.81j\sin(\mu+\nu)]}$$

$$= \frac{0.01}{\underbrace{[1 - 0.9(\cos\mu + \cos\nu) + 0.81\cos(\mu+\nu)]}_{a} + j\underbrace{[0.9(\sin\mu + \sin\nu) - 0.81\sin(\mu+\nu)]}_{b}}$$

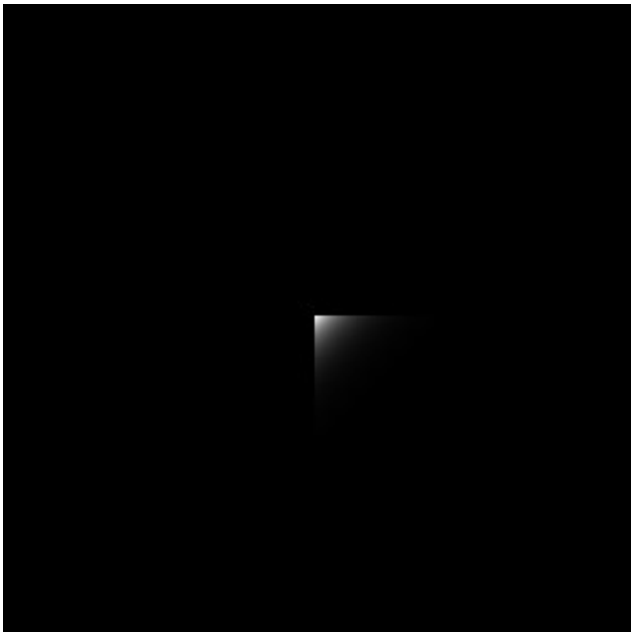
$$= \frac{0.01}{a + jb} = \frac{0.01}{a + jb} \cdot \frac{a - jb}{a - jb} = \frac{0.01(a - jb)}{a^2 + b^2} = k(a - jb) \; ; \; k = \frac{0.01}{a^2 + b^2}$$

$$|k(a - jb)| = \sqrt{(ka)^2 + (kb)^2} = k\sqrt{a^2 + b^2} \; ; \; a = 1 - 0.9(\cos\mu + \cos\nu) + 0.81\cos(\mu+\nu)$$
$$b = 0.9(\sin\mu + \sin\nu) - 0.81\sin(\mu+\nu)$$

2. Plot of $\left|H\left(e^{j\mu}, e^{j\nu}\right)\right|$



3. Image of the point spread function (PSF)



4. Filtered output color image, *filtered.tif*

5. Listing of C Code: *See Next Page*

```c

#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);
// initialize limitIntensity function
int limitIntensity(double value);
// initialize applyFilter function
void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img);

int main (int argc, char **argv)
{
    FILE *fp;
    struct TIFF_img input_img, color_img;

    if ( argc != 2 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
    }

    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
    }

    /* set up structure for output color image */
    /* Note that the type is 'c' rather than 'g' */
    get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );

    // declare and initialize integer to store the dimension of the point spread function
    int PSF_dim = 9;

```

```
49      // apply filter using applyFilter function as defined below main
50      applyFilter(&color_img, &input_img);
51
52      /* open color image file */
53      if ( ( fp = fopen ( "filtered.tif", "wb" ) ) == NULL ) {
54          fprintf ( stderr, "cannot open file color.tif\n");
55          exit ( 1 );
56      }
57
58      /* write color image */
59      if ( write_TIFF ( fp, &color_img ) ) {
60          fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
61          exit ( 1 );
62      }
63
64      /* close color image file */
65      fclose ( fp );
66
67      /* de-allocate space which was used for the images */
68      free_TIFF ( &(input_img) );
69      free_TIFF ( &(color_img) );
70
71      return(0);
72  }
73
74  void error(char *name)
75  {
76      printf("usage:  %s  image.tiff \n\n",name);
77      printf("this program reads in a 24-bit color TIFF image.\n");
78      printf("It then horizontally filters the green component, adds noise,
          \n");
79      printf("and writes out the result as an 8-bit image\n");
80      printf("with the name 'green.tiff'.\n");
81      printf("It also generates an 8-bit color image,\n");
82      printf("that swaps red and green components from the input image");
83      exit(1);
84  }
85
86  // limitIntensity function definition
87  int limitIntensity(double inputValue) {
88      // declare an integer variable newValue and initialize it to zero
89      int newValue = 0;
90      // if input value parameter is less than zero, assign new value to 0
91      if (inputValue < 0) {
92          newValue = 0;
93      }
94      // if input value parameter is greater than 255, assign new value to
          255
95      else if(inputValue > 255) {
```

```
 96          newValue = 255;
 97      }
 98      // otherwise, assign new value to the input value parameter re-cast as
         an integer
 99      else {
100          newValue = (int)inputValue;
101      }
102      return newValue;
103  }
104
105  // applyFilter function definition
106  void applyFilter(struct TIFF_img* output_img, struct TIFF_img* input_img)
       {
107      // declare and define image height and width based on input image TIFF
         struct methods
108      int img_height = input_img->height;
109      int img_width = input_img->width;
110      // define array of size three to store RGB information for each pixel
111      double plane[3];
112      // for each pixel
113      for (int i = 0; i < img_height; i++) {
114          for (int j = 0; j < img_width; j++) {
115              // for each plane in RGB pixel
116              for (int k = 0; k < 3; k++) {
117                  // assign to plane the term that will always exist
118                  plane[k] = 0.01 * input_img->color[k][i][j];
119                  if (i > 0) {
120                      // assign to plane the term that exists if i > 0
121                      plane[k] += 0.9 * (input_img->color[k][i - 1][j]);
122                  }
123                  if (j > 0) {
124                      // assign to plane the term that exists if j > 0
125                      plane[k] += 0.9 * (input_img->color[k][i][j - 1]);
126                  }
127                  if (i > 0 && j > 0) {
128                      // assign to plane the term that exists if i > 0 & j >
                         0
129                      plane[k] += - 0.81 * (input_img->color[k][i - 1][j -
                        1]);
130                  }
131                  // populate output image method for color after calling
                       limitIntensity function to ensure acceptable RGB values
132                  output_img->color[k][i][j] = limitIntensity(plane[k]);
133              }
134          }
135      }
136  }
```